IN THE SPECIFICATION:

Please replace the paragraph in the REFERENCE TO RELATED APPLICATIONS section on page 1 with the following amended paragraph:

The present application claims priority to and incorporates the following applications by reference: DYNAMIC SYMBOLIC LINK RESOLUTION, Prov. No. 60/157,728, filed on October 5, 1999; SNAPSHOT VIRTUAL TEMPLATING, Prov. No. ~~60/157,728~~ 60/157,729, filed on October 5, 1999; SNAPSHOT RESTORE OF APPLICATION CHAINS AND APPLICATIONS, Prov. No. 60/157,833, filed October 5, 1999; VIRTUAL RESOURCE-ID MAPPING, Prov. No. 60/157,727, filed on October 5, 1999; and VIRTUAL PORT MULTIPLEXING, Prov. No. 60/157,834 filed on October 5, 1999.

Please replace the paragraph beginning on page 5, line 15, as previously amended, with the following amended paragraph:

The virtual environment 110 is a layer that surrounds application(s) 208 and resides between the application and the operating system 206. Resource handles are abstracted to present a consistent view to the application although the actual system resource handles may change as an application is snapshot/restored more than once. The virtual environment also allows multiple applications to compete for the same resources where exclusion would normally prohibit such behavior to allow multiple snapshots to coexist without reconfiguration. Preload library 214 is an application library that interposes upon an application for the express purpose of intercepting and handling library ~~called~~ calls and system calls. Once the library has been preloaded it is attached to the process' address space. Preload library 214 interposes between application 208 and operating system 206. It is distinguished from kernel interposition in that it operates in "user mode" (i.e., non-kernel and non-privileged mode). Application 208 can make application programming interface (API) calls that modify the state of the application. These calls are made from the application 208 to the operating system API interfaces 210

via the application snapshot restore framework 200 or the preload library 214. The preload library can save the state of various resources by intercepting API interface calls and then saves the state at a pre-arranged memory location. When the process' memory is saved as part of the snapshot/restore mechanism, this state is saved since it resides in memory. The state as it is modified is saved to non-volatile storage (i.e. a file on disk). The preload library notify the snapshot/restore framework through one of its private interface.

Please replace the paragraph beginning on page 7, line 11 with the following amended paragraph:

Once all related processes are suspended, for each state (step 266) of each suspended process (step 264), the state is checked to see if it is virtualized (step 268 ~~264~~). A virtualized state is any process state that reflects a virtualized resource. If the state is virtualized, it is retrieved at step 270 ~~266~~; otherwise the non-virtualized state is retrieved at step 272.~~268~~ State retrieval is performed as described above by the snapshot driver 218 querying the application snapshot/restore framework 200, operating system API interfaces 210, and process management subsystem 216. If the state has changed since the last snapshot (step 274 ~~270~~), the new state is recorded (step 276). Control then loops to step 264 (through decision steps 278 and 280) and executes through the above sequence of steps until all states of all processes are checked. Once completed, control proceeds to step 282 (through decision steps 278 and 280) ~~278~~, the registered global state, such as semaphores, is removed. Registered global state is state that is not specifically associated with any one process (ie private state). Global state is usually exported (accessible) to all processes and its state can be modified (shared) by all processes. Control proceeds to step 284 ~~280~~, where the process is terminated. If there are remaining processes (step 286 ~~282~~), these are also terminated. This sequence of steps is concluded to create a snapshot image which is stored as a file and made available for transmission to another computer within public computer network 100 or private computer network 106.

Please replace the paragraph beginning on page 9, line 9 with the following amended paragraph:

In another aspect, the present invention provides a system, method, and computer program product for creating snapshot virtual application templates for the purpose of propagating a single application snapshot into multiple distinct instances. Snapshot virtual templates allow multiple application instances to use the same fixed resource ID ("RID") by making the resource ID virtual, privatizing the virtual RID, and dynamically mapping it to a unique system resource ID. A RID is the identifier assigned to represent a specific system resource and acts as a handle when referencing that system resource. Anonymous resources are resources that are read-only or functionally isolated from other applications. Anonymous resources are also shareable resources. An anonymous resource is a non-fixed resource allocated by the operating system and identified by a per-process handle. These are functionally-isolated since the operating system allocates it anonymously and one is as ~~food~~ good as another. Examples of this are non-fixed TCP ports or file descriptors. A resource is said to be network-wide unique if there can only be one instance of that resource with its corresponding identifier on computer network or subnetwork. An example of this is an network IP address (i.e. 10.1.1.1). Snapshot virtual templates allow snapshots to be described in a manner that separates shareable data from non-sharable data. Data is loosely defined to mean any system resource (memory, files, sockets, handles, etc.). When a snapshot is cloned from a virtual template, the common or shared data is used exactly as is, whereas the non-sharable data is either copied-on-write, multiplexed, virtualized, or customized-on-duplication. The present invention greatly reduces the required administrative setup per application instance. Snapshot virtual templating works by noting access to modified resources, fixed system IDs/keys and unique process-related identifies and automatically inserting a level of abstraction between these resources and the application. The resources contained in a snapshot virtual template can be dynamically redirected at restore time. Access to memory and storage is managed in a copy-on-write fashion. System resource handles are managed in a virtualize-on-allocate fashion or by a multiplex-on-access mechanism. Process-unique

resources are managed in a redirect-on-duplicate fashion. Rules may be defined through an application configurator that allows some degree of control over the creation of non-sharable data.

Please replace the paragraph beginning on page 10, line 19 with the following amended paragraph:

As used herein, non-sharable data refers to any resource that is modified and globally visible to other application instances ~~is non-sharable~~ (i.e. files). Process-related identifiers that are system-wide unique are also non-shareable since conflicts will arise if two instances use the same identifier at the same time (uniqueness is no longer preserved). References to unique resources by fixed handles (i.e. fixed TCP port numbers or IPC keys) are also not shareable. Memory pages that are specific to an application instance (i.e. the stack) are another example of a nonshareable resource. For illustrative purposes, examples of non-sharable data include application config files that must be different per application instance as well as modified application data files if the application is not written to run multiple copies simultaneously. Other examples include stack memory segments or heap segments may also be non-sharable data, shared memory keys that are a fixed value, usage of fixed well-known (to the application) TCP port numbers, and process identifiers (two distinct processes cannot share the same PID).